Welcome to IEEE Xplore®

- Home
- What Can I Access?
- Log-out

Tables of Contents

- Journals & Magazines
- Conference Proceedings
- Standards

Search

- By Author
- Basic
- Advanced

Member Services

- Join IEEE
- Establish IEEE Web Account
- Access the IEEE Member Digital Library

Your search matched **0** of **1011253** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or enteri new one in the text box.

'session state' <and> xml Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

---

Results:
No documents matched your query.

W3C®

NOTE-widl-970922

Web Interface Definition Language (WIDL)

Submitted to W3C 22 September 1997

This version:
http://www.w3.org/TR/NOTE-widl-970922
Latest Version:
http://www.w3.org/TR/NOTE-widl
Authors:
Phillip Merrick, webMethods, phillip@webMethods.com
Charles Allen, webMethods, caallen@webMethods.com

Status of this Document

This document is a NOTE made available by the W3 Consortium for discussion only. This indicates no endorsement of its content, nor that the Consortium has, is, or will be allocating any resources to the issues addressed by the NOTE.

This document is a submission to W3C from webMethods, Inc.. Please see Acknowledged Submissions to W3C regarding its disposition.

Abstract

This document provides the specification for the Web Interface Definition Language (WIDL), a metalanguage that implements a service-based architecture over the document-based resources of the World Wide Web. WIDL is an application of the eXtensible Markup Language (XML); it allows interactions with Web servers to be defined as functional interfaces that can be accessed by remote systems over standard Web protocols, and provides the structure necessary for generating client code in languages such as Java, C/C++, COBOL, and Visual Basic. WIDL enables a practical and cost-effective means for diverse systems to be rapidly integrated across corporate intranets, extranets, and the Internet.

Contents

---

1. Introduction

The World Wide Web is providing millions of end-users access to ever-increasing volumes of information. While the Web initially served browsers static documents, the resources of legacy systems, relational databases and multi-tier applications have all been made available to the Web browser to provide corporate users with interactive information resources including financial services, inventory management, on-line purchasing and package tracking.

While the Web has achieved the extraordinary feat of providing ubiquitous accessibility to end-users, it has in many cases reinforced manual inefficiencies in business processes as repetitive tasks are required to transcribe or copy and paste data from browser windows into desktop and corporate applications. This is as true of Web data provided by remote business units and external (i.e. partner or supplier) organizations as it is of Web data accessible from both

public and subscription based Web sites. The problem of direct access to Web data from within business applications has been largely ignored.

The purpose of the Web Interface Definition Language (WIDL) is to enable automation of all interactions with HTML/XML documents and forms, providing a general method of representing request/response interactions over standard Web protocols, and allowing the Web to be utilized as a universal integration platform.

A central feature of WIDL is that programmatic interfaces can be *defined* and *managed* for data (HTML, XML or text files) and services (CGI-bin, database, or other back end systems) that are not under the direct control of programs that require such access. WIDL definitions can be co-located with client programs, centrally managed in a client/server architecture, or referenced directly from HTML/XML documents.

WIDL defintions provide a mapping between Web resources and applications written in conventional programming languages such as C/C++, COBOL, Visual Basic, Java, JavaScript, etc., enabling automatic and structured Web access by compatible client programs, including mainstream business applications, desktop applications, applets, Web agents, and server-side Web programs (CGI, etc.).

Automatic means that complex interactions with Web servers do not require human intervention; programs can request Web data and services by making local calls to functions which encapsulate standard Web access protocols and utilize WIDL definitions to provide naming services, change management, error handling, condition processing and intelligent data binding.

Structured means that Web data and services are described as interfaces with well defined input and output variables.

Standard Web access protocols means HTTP and HTTPS.

Compatible means any program that both utilizes WIDL definitions to define the location of Web services and the structure of data that is returned by standard HTTP and HTTPS requests, and allows WIDL definitions to be managed locally, centrally, or by individual service providers.

WIDL describes business objects on the Web, providing the basis for a common API across Web servers, legacy systems, databases, and middleware infrastructures, and effectively transforming the Web from an access medium into an integration platform.

This document provides a complete description of the Web Interface

Definition Language (WIDL).

2. Benefits of WIDL

A major part of the value of an Interface Defintion Language (IDL) is that it can define services offered by applications in an abstract but highly usable fashion. WIDL brings to the Web many of the features of IDL concepts that have been implemented in distributed computing and transaction processing platforms including DCE, and CORBA.

2.1 Business-to-Business Integration

WIDL makes it easy for organizations to automate business transactions with customers and suppliers. WIDL describes and automates interactions with services hosted by Web servers on intranets, extranets and the Internet; it transforms the Web into a standard integration platform and provides a universal API for all Web-enabled systems.

Using HTML, XML, HTTP and HTTPS as corporate standards glue, WIDL requires only that target systems be Web-enabled. There are hundreds of products in the market today which Web-enable existing systems, from mainframes to client/server applications. The use of standard Web technologies empowers various IT departments to make independent technology selections. This has the effect of lowering both the technical and 'political' barriers that have typically derailed cross-organizational integration projects.

A number of analysts have already warned that proprietary e-commerce platforms could lock suppliers into relationships by forcing them to integrate their systems with one infrastructure for business-to-business integration, making it costly for them to switch to or integrate with other partners who have selected alternate e-commerce platforms. Buyer-supplier integration issues involve many-to-many relationships, and demand a standard platform for functional integration and data exchange.

A service defined by WIDL is equivalent to a function call in standard programming languages. At the highest level, WIDL files describe the locations (URLs) of services, input parameters to be submitted (via Get or Post methods) to each service, conditions for successful processing, and output parameters to be returned by each service.

WIDL provides the following features:

- A browser is not required to drive Web applications
- WIDL definitions are dynamically interpreted and can be centrally managed
- Client applications are insulated from changes in service locations and data extraction methods
- Developers are insulated from network programming concerns
- Application resources can be integrated through firewalls and proxies

WIDL can be used to describe interfaces and services for:

- Static documents (HTML, XML, and plain text files)
- Dynamically generated documents (HTML, XML, and plain text files)
- HTML forms
- URL directory structures

WIDL can be used:

- to automate interactions with Web servers
- for both on-demand and scheduled extraction of targeted Web data
- to aggregate data from a number of Web sources
- to chain services across multiple Web sites
- to rapidly integrate Web resources with traditional application development languages and environments

WIDL has the ability to specify conditions for successful processing, and error messages to be returned to calling programs. Conditions further enable services to be defined that span multiple documents.

2.2 Change Management

One of WIDL's most significant benefits is its ability to insulate client programs from changes in the format and location of Web documents. Unlike the way CORBA and DCE IDL are normally used, WIDL is interpreted at runtime; as a result, service URLs, object references in variables, definitions of document regions, success/failure conditions, and directives for service chaining can all be administered without requiring modification of client code. This usage model supports application-to-application linkages that are far more robust and maintainable than if they were coded by hand.

There are three models for WIDL management:

- client side - where WIDL files are co-located with a client program
- naming service - where WIDL definitions are centrally managed and referenced via directory services, i.e. LDAP
- server side - where WIDL files are referenced by, co-located with, or embedded within Web documents.

WIDL does not require that existing Web resources be modified in any way. Flexible management models allow organizations to describe and integrate Web sites that are uncontrolled, as well as to provide their business partners with interfaces to services that are controlled. The ability to seamlessly migrate from independent to shared management eases the transition from informal to formal business-to-business integration.

2.3 Language Bindings

The primary purpose of WIDL is integration of Web resources with corporate business applications. In much the same way that DCE or CORBA IDL is used to generate code fragments, or 'stubs', to be included in application development projects, WIDL provides the structure necessary for generating client code in languages such as C/C++, Java, COBOL, and Visual Basic. Developers can thus be insulated from the need to understand both HTML/XML parsing and Web protocols. This capability enables the existing skills of innumerable programmers to be rapidly leveraged in the utilization of Web based resources.

---

3. Object Model References

Many of the features of WIDL require a capability to reliably identify and extract specific data elements from Web documents. Various mechanisms for accessing elements of HTML and/or XML documents have been defined, such as the Javascript Page Object Model, the Document Object Model, and XLL XPointers. WIDL does not define or determine a mechanism for accessing document data, but rather allows an object model referencing mechanism to be specified on a per-interface basis.

The following capabilities are desirable for accessing elements of Web documents:

- HTML Parsing
- XML Parsing
- Text Pattern Matching

Object referencing mechanisms would ideally support both parsing and pattern matching. Pattern matching extracts data based on regular expressions, and is well suited to raw text files and poorly constructed HTML documents. Parsing, on the other hand, recovers document structure and exposes relationships between document objects, enabling elements of a document to be accessed with an object model.

4. WIDL Examples

The following example illustrates the use of WIDL to define a package tracking service for generic Shipping. By allowing a WIDL definition to reference a 'Template' WIDL definition, a general class of shipping services can be defined. 'FoobarShipping' is one implementation of the 'Shipping' interface.

```
<WIDL NAME="genericShipping" TEMPLATE="Shipping"
      BASEURL="http://www.shipping.com" VERSION="2.0">

<SERVICE NAME="TrackPackage" METHOD="Get"
         URL="/cgi-bin/track_package"
         INPUT="TrackInput" OUTPUT="TrackOutput" />

<BINDING NAME="TrackInput" TYPE="INPUT">
   <VARIABLE NAME="TrackingNum" TYPE="String" FORMNAME="trk_num" />
   <VARIABLE NAME="DestCountry" TYPE="String" FORMNAME="dest_cntry" />
   <VARIABLE NAME="ShipDate" TYPE="String" FORMNAME="ship_date" />
</BINDING>

<BINDING NAME="TrackOutput" TYPE="OUTPUT">
   <CONDITION TYPE="Failure" REFERENCE="doc.title[0].text"
              MATCH="Warning Form" REASONREF="doc.p[0].text" />
   <CONDITION TYPE="Success" REFERENCE="doc.title[0].text"
              MATCH="Foobar Airbill:*" REASONREF="doc.p[1].value" />
   <VARIABLE NAME="disposition" TYPE="String" REFERENCE="doc.h[3].value" />
   <VARIABLE NAME="deliveredOn" TYPE="String" REFERENCE="doc.h[5].value" />
   <VARIABLE NAME="deliveredTo" TYPE="String" REFERENCE="doc.h[7].value" />
</BINDING>

</WIDL>
```

In this example, the values defined in the 'TrackInput' binding get passed via HTTP Get as name-value pairs to a service residing at 'http://www.shipping.com/cgi-bin/track_package'. Object References are used in the 'TrackOutput' binding to a) check for successful completion of the service, and b) extract data elements from the document returned by the HTTP request.

4.1 Interface Templates

WIDL enables common interfaces to services provided by multiple sites. Templates allow the specification of interfaces, implementations of which may be available from multiple sources. A shipping template defines a functional interface for shipping services; various implementations can be provided for Federal Express, UPS, and DHL.

```
<WIDL TEMPLATE="Shipping">
```

```
    <SERVICE NAME="packageTrack" ... />
    <SERVICE NAME="schedulePickup" ... />

  ...
```

4.2 Service Timeouts

Service definitions support TIMEOUTS and RETRIES to handle situations when a Web server is responding intermittently. If a service does not complete within the specified number of seconds, it is tried again up to RETRIES times after which it fails.

```
  <SERVICE NAME="schedulePickup" METHOD="POST"
           URL="http://www.fooShipping.com"
           INPUT="pickupInput" OUTPUT="pickupOutput"
           TIMEOUT="5" RETRIES="5" />
   ...
```

4.3 Bindings

'Input' and 'Output' bindings specify the input and output variables of a particular service. Input bindings define the name-value pairs to be passed via Get or Post methods to a Web-based application. Output bindings use object references to identify and extract data elements from documents returned by HTTP requests.

```
    <BINDING NAME="TrackInput" TYPE="INPUT">
    ...
    <BINDING NAME="TrackOutput" TYPE="OUTPUT">
    ...
```

4.4 Form Names

The FORMNAME attribute of a variable declaration enables the parameters of a service to be re-named. This feature is useful for defining multiple implementations of a service which require a common interface, yet must pass the proper name-value pairs to individual Web-sites where the services are hosted.

```
     <BINDING NAME="TrackInput" TYPE="INPUT">
       <VARIABLE NAME="TrackingNum" TYPE="String" FORMNAME="trk_num" />
    ...
```

4.5 HTTP Headers

The values of variables declared as USAGE="Header" get passed as part of an HTTP header and are not included in the name-value pairs submitted to back-end services (i.e. CGI script) via Get or Post. Variable values may also be hardcoded by providing a Value.

```
<BINDING NAME="anInput" TYPE="INPUT">
   <VARIABLE NAME="REFERRER" TYPE="String"
             VALUE="http://www.company.com"  USAGE="HEADER" />
...
```

4.6 Internal Variables

The values of variables declared as USAGE="Internal" are accessible within WIDL declarations and are not included in the name-value pairs submitted to back-end services (i.e. CGI script) via Get or Post. In this example the value an input variable 'state' is used to complete the URL for the service 'AutoLoan'. Internal variable enable URL directory structures to be interactively 'queried'.

```
<SERVICE NAME="AutoLoan" Method="Get"
         URL="http://www.autoloan.com/%state%.html"
         INPUT="AutoLoanInput" OUTPUT="AutoLoanOutput" />

<BINDING NAME="AutoLoanInput" TYPE="INPUT">
   <VARIABLE NAME="state" TYPE="String" USAGE="INTERNAL" />
...
```

4.7 Regions

The REGION element defines an area of a document by specifying START and END object references. Named regions permit the extraction of data elements relative to other features of a document. Regions are addressed using object references that begin with the region name.

```
<BINDING NAME="NewsOut" TYPE="OUTPUT">
   <REGION NAME="tops" START="doc.p[3]" END="doc.h[4]" />
   <VARIABLE NAME=stories TYPE="String[]" REFERENCE="tops.a[].text" />
   <VARIABLE NAME="links" TYPE="String[]" REFERENCE="tops.a[].href" />
   ...
```

4.8 Conditions

Conditions define 'success' and 'failure' states for output bindings, and determine whether a binding attempt should be retried in the case of a 'server busy' error:

```
<BINDING NAME="AutoLoanOutput" TYPE="OUTPUT">
   <CONDITION TYPE="FAILURE" REASONTEXT="State not found" />
...

<BINDING NAME="TrackOutput" TYPE="OUTPUT">
   <CONDITION TYPE="SUCCESS" REFERENCE="doc.title[0].text"
              MATCH="Shipping Airbill:*"
              REASONREF="doc.p[1].value" />
...

<BINDING NAME="getPrice" TYPE="OUTPUT">
   <CONDITION TYPE="RETRY" REFERENCE="doc.headings[0].text"
```

```
                MATCH="*Server Busy*"
                WAIT="5" RETRIES="4" />
    ...
```

Conditions can apply to a binding as a whole, or to a specific object reference. Conditions can define error messages to be returned as the value of the service; error messages can be a literal, or can be extracted from the returned document.

This mechanism can handle not only unexpected errors, but also can be used to map application-level errors from the back-end program (i.e. responses resulting from invalid or missing input values).

4.9 Multiple Bindings

Conditions can direct a service to attempt an alternate binding for the extraction of output values:

```
<BINDING NAME="getPrice" TYPE="Output">
   <CONDITION TYPE="FAILURE" REBIND="shirtPrice" />
...
<BINDING NAME="shirtPrice" TYPE="Output">
...
```

Multiple bindings are useful in situations where the documents returned by a back-end program are dependent upon the input criteria that was submitted in the HTTP request. For example, a retail Web site may return a document with a different structure for an SKU depending on whether the item requested is a shirt, a tie, or trousers. The use of multiple bindings allows a condition to determine the appropriate binding for extracting the desired data. Must refer to a binding that is defined in the same WIDL interface.

4.10 Service Chains

Conditions can direct a service to initiate a service chain, in which case the name-value pairs of an output binding are passed into a second service. The name-value pairs must match the variable names in the input binding of the second service for the service-chain to succeed.

```
<Binding Name="productSearchOutput" Type="Output">
  <Condition Type="Success" Service="ExtractPrices" />
...
```

Service chains can be used with Web-based e-commerce systems when it is necessary to invoke multiple services in sequence to complete a purchase.

5. WIDL Reference

The Web Interface Definition Language (WIDL) is an application of the eXtensible Markup Language (XML); its definition consists of the various XML elements defined in this section.

Major Elements

WIDL - defines an interface
SERVICE - child of WIDL - defines a service
BINDING - child of WIDL - defines input and output parameters for services

Minor Elements

VARIABLE - child of BINDING - defines a variable within a binding
CONDITION - child of BINDING - defines a condition within a binding
REGION - child of BINDING - defines a area within a document

The following sections define the elements of WIDL.

5.1 WIDL Element

<WIDL> is the parent element for the Web Interface Definition Language; it defines an interface. Interfaces are groupings of related services and bindings. The following are attributes of the <WIDL> element:

Attribute	Description	Type	#	Default
NAME	Establishes a name for an interface. The interface name is used in conjunction with a service name for naming or directory services.	String	Exactly One	
VERSION	Identifies the version of WIDL.	String	0 or 1	"2.0"
TEMPLATE	WIDL enables common interfaces to services provided by multiple vendors. A shipping template defines a functional interface for shipping services; various implementations can be	URI	0 or 1	

	provided for Federal Express, UPS, and DHL.			
BASEURL	BASEURL is similar to the <BASE HREF=""> statement in HTML. Some of the services within a given WIDL may be hosted from the same Base URL. If BASEURL is defined, the URL for various services can be defined relative to BASEURL. This feature is useful for replicated sites which can be addressed by changing only the BASEURL, instead of the URL for each service.	URI	0 or 1	
OBJMODEL	Specifies an object model to be used for extracting data elements from HTML and XML documents. Object models are the result of parsing HTML or XML documents. The use of object models is central to the functionality of WIDL. Object References are used in <VARIABLE/>, <CONDITION/> and <REGION/> elements.	String	0 or 1	wmdom

5.2 Service Element

The <SERVICE/> element describes a request/response interaction with a Web server. Web servers use Get and Post methods to return documents and invoke CGI scripts and services via NSAPI, ISAPI, or other back-end Web server programs. Web servers typically take a set of input parameters, perform some processing, then return a static or dynamically generated HTML, XML or text document.

The attributes of the <SERVICE/> element map an abstract service name to an actual URL, specify the HTTP method to be used to access the service, and designate 'bindings' for input and output parameters.

Attribute	Description	Type	#	Default
NAME	Establishes a name for a service. The service name is used in conjunction with an interface name for naming or directory services	String	Exactly One	

URL	Specifies the Uniform Resource Locator (URL) for the target document. A service URL can be either a fully qualified URL or a partial URL that is relative to the BaseURL provided as an attribute of the <WIDL> element.	URI	Exactly One	
METHOD	specifies the HTTP method ("Get" or "Post") to be used to access the service.	String	Exactly One	"Get"
INPUT	Designates the <Binding> to be used to define the input parameters for programs that call the service. The specified name must be that of a <BINDING> contained within the same <WIDL> as the service.	String	0 or 1	
OUTPUT	Designates the <Binding> to be used to define the output parameters for programs that call the service. The specified name must be that of a <BINDING> contained within the same <WIDL> as the service.	String	0 or 1	
AUTHUSER	Establishes the Username to be submitted for HTTP Authentication.	String	0 or 1	
AUTHPASS	Establishes the Password to be submitted for HTTP Authentication.	String	0 or 1	
TIMEOUT	If the service does not complete within the specified number of seconds, it is tried again up to RETRIES times after which it fails.	String	0 or 1	
RETRIES	Number of times to retry the service before failing.	String	0 or 1	

5.3 Binding Element

The <BINDING> element defines input and output variables for a

service. Input bindings describe the data submitted via Get or Post to a Web server; for example, the input fields in an HTML form. Static HTML document do not require input variables. Output bindings describe which data elements are to be mapped from the output document and returned as a result of an HTTP request to a Web server with the given input variables. In most cases an output binding will map only a subset of the available data elements in the output document.

Attribute	Description	Type	#	Default
NAME	Establishes a name for a binding. The name is used in <SERVICE/> definitions and in <CONDITION/> statements that initiate service chains.	String	Exactly One	
TYPE	Specifies whether a binding defines input or output variables.	String	Exactly One	"Output"

5.4 Variable Element

The <VARIABLE/> element is used to describe both input and output binding parameters. Different attributes are used depending on the type of parameter being described.

Attribute Name	Description	Type	#	Default
NAME	Identifies both the program variable and the VARIABLE definition itself.	String	Exactly One	
FORMNAME	*BINDING TYPE="Input":* Specifies the variable name to be submitted via Get or Post methods. Obscure back-end variables can be given names that are more meaningful in the context of the service described by WIDL. Used in conjunction with WIDL Templates, FORMNAME permits the mapping of a single variable name across multiple service implementations.	String	0 or 1	
	Specifies both the data type			

TYPE	and dimension of the variable.	String	0 or 1	
USAGE	The default usage of variables is for specification of input and output parameters. Variables can also be used internally within WIDL, as well as to pass header information in an HTTP request. For instance, using internal variables a portion of a service's URL or a pattern for matching within an object reference can be specified as a variable that is part of an input binding.	String	Exactly One	Default
VALUE	Designates a value to be assigned to the variable in HTTP transactions. For input variables this has the effect of rendering the variable invisible to calling programs, i.e. the specified value is submitted to the Web server without requiring an input from calling programs. For output variables this has the effect of hard-coding the value returned when the service is invoked.	String	0 or 1	
REFERENCE	*BINDING TYPE="Output":* Any valid object reference defined by the specified Object Model. Identifies a property (typically the value) of an HTML document object to be assigned to the associated program output variable. If the identified object is not present in the output document, or the specified property is not valid for the object, null is assigned into the program output	String	0 or 1	

	variable.			
NULLOK	*BINDING TYPE="Output":* Overrides the implicit condition that all output variables return a non-null value.	String	0 or 1	"False"

5.5 Condition Element

The <CONDITION/> element is used in output bindings to specify success and failure conditions for the binding of data to be returned to calling programs. Conditions enable branching logic within service definitions; they are used to attempt alternate bindings when initial bindings fail and to initiate service chains, whereby the output variables from one service are passed as name-value pairs into the input bindings of a second service. Conditions also define error messages returned to calling programs when services fail.

Attribute Name	Description	Type	#	Default
TYPE	Specifies whether a condition is checking for the 'Success' or the 'Failure' of a binding attempt, or whether a binding attempt should be retried in the case of a 'server busy' error. Any variable that returns a NULL value will cause the entire binding to fail, unless the NULLOK attribute of that variable has been set to true. Conditions can catch the success or failure of either a specific object reference or of an entire binding. In the case where a condition initiates a service chain, it is important that all variables bind properly.	String	0 or 1	"Success"
	Specifies an object reference which extracts			

REFERENCE	data from the HTML or XML document returned as the result of a service invocation. The Reference attribute for conditions is equivalent to the Reference attribute used in variable definitions. Identifies the document object property that will be compared with the text pattern specified by the Match attribute.	String	Exactly One	
MATCH	Specifies a text pattern that will be compared with the object property referenced by the Reference attribute.	String	Exactly One	
REBIND	Specifies an alternate output binding. Typically a failure condition indicates that the document returned cannot be bound properly. Rebind redirects the binding attempt. The use of REBIND allows a conditions to determine the appropriate binding for extracting the desired data. Must refer to a binding that is defined in the same WIDL interface.	String	0 or 1	
SERVICE	Specifies a service to invoke with the results of an output binding. Aside from the obvious benefit of chaining services to further automate the tasks that can be encapsulated for client programs, there are many cases when target documents can only be retrieved after visiting several Web	String	0 or 1	

	pages in succession. In some instances cookies are issued by an entry page that must be visited prior to interacting with HTML forms; in others, URLs are dynamically generated from databases for specific user identities.			
REASONREF	Reference to an object's attribute to be returned as an error message when a service fails.	String	0 or 1	
REASONTEXT	The text to be returned as an error message when a service fails.	String	0 or 1	
WAIT	Number of seconds to wait before re-trying retrieval of a document after a server has returned a 'service busy' error.	String	0 or 1	
RETRIES	Number of times to retry the service before failing.	String	0 or 1	

5.6 REGION Element

The <REGION/> element is used in output bindings to define targeted sub-regions of a document. Regions permit the extraction of data elements relative to other features of a document. Regions are critical for poorly designed documents where it is otherwise impossible to differentiate between desired data elements (for instance story links on a news page) and elements that also match the search criteria.

Attribute	Description	Type	#	Default
NAME	Specifies the name for a region. This name can then be used as the root of an object reference.	String	Exactly One	
START	An object reference that determines the beginning of a region.	String	Exactly One	

END	An object reference that determines the end of a region.	String	Exactly One	

6. Conclusion

Web technology is strong on interactivity, but low on automation. Electronic commerce on the Web is primarily driven manually via a browser. In order to achieve business-to-business integration organizations have resorted to proprietary protocols. The many-to-many nature of Web commerce demands a standard for automated integration.

This proposal defines the infrastructure necessary for Web resources to be described as functional interfaces that can be invoked directly from business applications written in languages such as Java, C/C++, COBOL, and Visual Basic. By capturing details such as input parameters, service URLs, and data extraction methods for output parameters, WIDL enables automation of interactions normally performed manually via a browser.

---

7. References

- XML W3C Working Draft by Tim Bray, Jean Paoli, C.M. Sperberg-McQueen
- Extensible Markup Language (XML): Part2. Linking by Tim Bray, Steve DeRose
- Document Object Model W3C Working Group
- XML, Java, and the Future of the Web by Jon Bosak
- Automating the Web with WIDL by Charles Allen

Appendix A - The complete WIDL DTD

```
<!ELEMENT WIDL ( SERVICE | BINDING )* >
<!ATTLIST WIDL
     NAME       CDATA   #IMPLIED
     VERSION (1.0 | 2.0 | ...) "2.0"
     TEMPLATE   CDATA   #IMPLIED
     BASEURL    CDATA   #IMPLIED
     OBJMODEL (wmdom | ...) "wmdom"
>

<!ELEMENT SERVICE EMPTY>
<!ATTLIST SERVICE
     NAME       CDATA   #REQUIRED
     URL        CDATA   #REQUIRED
```

```
      METHOD (Get | Post) "Get"
      INPUT       CDATA   #IMPLIED
      OUTPUT      CDATA   #IMPLIED
      AUTHUSER    CDATA   #IMPLIED
      AUTHPASS    CDATA   #IMPLIED
      TIMEOUT     CDATA   #IMPLIED
      RETRIES     CDATA   #IMPLIED
>

<!ELEMENT BINDING ( VARIABLE | CONDITION | REGION )* >
<!ATTLIST BINDING
      NAME        CDATA   #REQUIRED
      TYPE (Input | Output) "Output"
>

<!ELEMENT VARIABLE EMPTY>
<!ATTLIST VARIABLE
      NAME        CDATA   #REQUIRED
      FORMNAME    CDATA   #IMPLIED
      TYPE (String | String[] | String[][]) "String"
      USAGE (Default | Header | Internal) "Function"
      REFERENCE   CDATA   #IMPLIED
      VALUE       CDATA   #IMPLIED
      MASK        CDATA   #IMPLIED
      NULLOK              #BOOLEAN
>

<!ELEMENT CONDITION EMPTY>
<!ATTLIST CONDITION
      TYPE  (Success | Failure | Retry) "Success"
      REF         CDATA   #REQUIRED
      MATCH       CDATA   #REQUIRED
      REBIND      CDATA   #IMPLIED
      SERVICE     CDATA   #IMPLIED
      REASONREF   CDATA   #IMPLIED
      REASONTEXT  CDATA   #IMPLIED
      WAIT        CDATA   #IMPLIED
      RETRIES     CDATA   #IMPLIED
>

<!ELEMENT REGION EMPTY>
<!ATTLIST REGION
      NAME        CDATA   #REQUIRED
      START       CDATA   #REQUIRED
      END         CDATA   #REQUIRED
>
```

NOTE-WAP-19981030

WAP Forum - W3C Cooperation White Paper

W3C Note 30 October 1998

This version
http://www.w3.org/TR/1998/NOTE-WAP-19981030
Latest version
http://www.w3.org/TR/NOTE-WAP
Editors:
Johan Hjelm, W3C / Ericsson
Bruce Martin, WAP Forum / Unwired Planet
Peter King, WAP Forum / Unwired Planet

Status of this Document

Version 1.1, September 28, 1998

This document is a NOTE made available by the W3 Consortium for discussion only. This indicates no endorsement of its content, nor that the Consortium has, is, or will be allocating any resources to the issues addressed by this NOTE.

Abstract

This paper outlines possible areas of cooperation between the WAP Forum and the World Wide Web Consortium (W3C).

The WAP Forum is dedicated to enabling advanced services and applications on mobile wireless devices, such as cellular telephones. The W3C is dedicated to leading and advancing the development of the World Wide Web. This document describes the problem area of mobile access to

information on the web, which is common to the two groups.

While the WAP Forum and the W3C have different organizational goals, we share goals for the future of the global information space. We also want to avoid unnecessary divergence between the recommendations and standards of the two organizations.

Direct overlaps in future development occur in the areas of intelligent proxies and protocol design; of XML applications; and in content adaption, e.g through the use of vector graphics and style sheets. Instead of developing diverging sets of solutions, it is the intent of both groups to find common solutions.

1.0 Introduction

1.1 W3C Background

The World Wide Web Consortium (W3C) was founded in 1994 by Tim Berners-Lee, the inventor of the World Wide Web. The web had then outgrown the European Centre for Nuclear Physics Research, CERN, where the web was developed.

From the beginning, Tim Berners-Lee has been the director of the consortium, and has been firmly committed to developing a neutral, open forum for the evolution of web technology.

Today, the W3C has three locations (at MIT in the USA; at INRIA in France; and at Keio University in Japan). The consortium has more than 270 members from industry and academia, and provides a vendor-neutral forum for its members to address web-related issues. Working with its staff and the global web community, the consortium aims to produce free, interoperable specifications; like its partner standards body, the Internet Engineering Task Force (IETF), the W3C is committed to backing its specifications by sample code. Funding from membership dues, public research funds, and external contracts underwrite these efforts. The work

in the W3C follow a well-documented process, continuously involving representatives of the member companies in its efforts. More information can be found at the consortium web site.

In the autumn of 1997, it became evident that there was a considerable interest among the members of the W3C for access to the web via mobile and wireless devices. This area appeared to entail several aspects which constrained the usage in manners not considered in the current recommendations.

A workshop on mobile web access was organised in April of 1998 in Tokyo to address these topics, leading to the formation of a mobile access interest group that is chartered with the investigation of the impact of mobile access on the specifications and recommendations of the W3C.

Simultaneously, the W3C was approached by the WAP Forum, and a meeting was set up in June 1998. The results were very positive, and the meeting decided to create this joint white paper, outlining areas of potential cooperation between the two groups.

1.2 WAP Forum Background

The Wireless Application Protocol (WAP) Forum is an industry group dedicated to the goal of enabling sophisticated telephony and information services on hand-held wireless devices such as mobile telephones, pagers, personal digital assistants (PDAs) and other wireless terminals. Recognizing the value and utility of the World Wide Web architecture, the WAP Forum has chosen to align certain components of its technology very tightly with the Internet and the WWW. The WAP specifications extend and leverage mobile networking technologies (such as digital data networking standards) and Internet technologies (such as IP, HTTP, XML, URLs, scripting and other content formats).

The WAP specification initiative began in June 1997 and the WAP Forum was founded in December 1997. The WAP Forum has drafted a global

wireless protocol specification for all wireless networks and will contribute it to appropriate industry and standards bodies. WAP will enable manufacturers, network operators, content providers and application developers to offer compatible products and secure services on all devices and networks, resulting in greater economies of scale and universal access to information. WAP Forum membership is open to all industry participants.

The objectives of the WAP Forum are:

To bring Internet content and advanced data services to digital cellular phones and other wireless terminals.
To create a global wireless protocol specification that will work across different wireless network technologies.
To enable the creation of content and applications that scale across a very wide range of wireless bearer networks and wireless device types.
To embrace and extend existing standards and technology wherever appropriate.

More information on the WAP Forum can be found at the Wap Forum Web server.

In keeping with its goals, the WAP Forum approached the W3C regarding collaboration in the area of WWW technologies in the wireless area.

2. Goals for the cooperation

2.1 Short-Term Goals

Bring Internet and WWW technologies to digital cellular phones and other wireless terminals, i.e., adapting the Web architecture to the wireless environment.
Establish productive working relationships between the W3C and WAP Forum in the areas where common organizational goals exist.

Reduce overlapping technical work between the W3C and WAP Forum.
Cross-reference technical specifications.
Joint test-bed and protocol validation work.

2.2 Long-Term Goals

Work toward a unified information space.
Work toward common standards and technologies.
Enable the delivery of sophisticated information and services to mobile wireless terminals.

3. The Technical Problem: Wireless System Requirements on Information Retrieval

Providing Internet and WWW services on a wireless data network presents many challenges. Most of the technology developed for the Internet has been designed for desktop and larger computers supporting medium to high bandwidth connectivity over generally reliable data networks.

Mass-market, hand-held wireless devices present a more constrained computing environment compared to desktop computers. Because of fundamental limitations of power and form factor, mass-market handheld devices tend to have:

Less powerful CPUs
Less memory (ROM and RAM)
Restricted power consumption
Smaller displays
Different input devices (e.g., a phone keypad, voice input, etc.)

Similarly, wireless data networks present a more constrained communication environment compared to wired networks. Because of fundamental limitations of power, available spectrum, and mobility, wireless data networks tend to have:

Less bandwidth than traditional networks
More latency than traditional networks
Less connection stability than other network technologies
Less predictable availability

Mobile networks are growing in complexity and the cost of providing new value-added services to wireless users is increasing. In order to meet the requirements of mobile network operators, solutions must be:

Interoperable - terminals from different manufacturers communicate with services in the mobile network.
Scalable - mobile network operators are able to scale services to customer needs.
Efficient - provides quality of service suited to the behaviour and characteristics of the mobile network; provide for maximum users for a given network configuration
Reliable - provides a consistent and predictable platform for deploying services.
Secure - enables services to be extended over potentially unprotected mobile networks while still preserving the integrity of user data; protects the devices and services from security problems such as denial of service.

3.1 Bearer Limitations

Wireless network bearers operate under several fundamental constraints, which place restrictions on the type of protocols and applications offered over the network:

Power consumption.
As bandwidth increases, power consumption increases. In a mobile device, this reduces battery life.

Cellular network economics.
Mobile networks are typically based on a cellular architecture. Cells are a resource shared by all mobile terminals in a geographic area,

and typically have a fixed amount of bandwidth to be shared among all users. This characteristic rewards efficient use of bandwidth, as a means of reducing the overall cost of the network infrastructure.

Latency.

The mobile wireless environment is characterized by a very wide range of network latency, ranging from sub-second round-trip communication time up to many tens of seconds. In addition, network latency can be highly variable, depending on the current radio transmission characteristics (e.g., in a tunnel or off network) and the network loading in a particular area. Latency is further increased by routing, error correction and congestion-avoidance characteristics of a particular network.

Bandwidth.

The mobile wireless environment is characterized by a very wide range of network characteristics, and typically has far less bandwidth available than a wireline environment. In addition, the economics of the wireless environment encourage the conservation of bandwidth to achieve greater density of subscribers.

3.2 Device Limitations

Wireless devices operate under a set of physical limitations, imposed by their mobility and form factor:

Limited power.

Any personal, or "hand held" mobile device will have a very limited power reserve, due to existing battery technology. This reduces available computational resources, transmission bandwidth, etc.

Size:

many mobile wireless are very small (hand-held).

Mobile wireless devices are characterized by a different set of user interface constraints than a personal computer. To enable a consistent application programming model, a very wide range of content scalability is required. In practice, a significant amount of the current WWW content is

unsuitable for use on hand-held wireless devices. Problems include:

Output scalability.
: Existing content is designed for viewing on PC screens, whereas mobile devices will have a wide range of visual display sizes, formatting and other characteristics. In the near future, this will include voice-only output.

Input scalability.
: Mobile devices feature a wide range of input models, including numeric keypad, very few or no programmable soft keys, etc. In the near future, this will include voice-only input.

3.3 Use Case Limitations

Many wireless devices, for example cellular phones and pagers, are consumer devices. These devices are used in a wide variety of environments and under a wide range of use scenarios. For example:

Simple user interfaces:
: many mobile devices, in particular, cellular telephones, are mass-market consumer-oriented devices. Their user interface must be extremely simple and easy to use.

Single-purpose devices:
: the goal and purpose of most mobile devices is very focused (e.g., voice communication). This is in contrast with the general-purpose tool-oriented nature of a personal computer. This motivates a very specific set of use cases, with very simple and focused behavior. For example, "place a voice call" or "find the nearest ATM."

Hands-free, "heads-up" operation:
: many mobile devices are used in environments where the user should not be unnecessarily distracted (e.g., driving and talking).

4. Future Development from the W3C in this Field

The World Wide Web Consortium has as its motto "Leading the Web to its

full potential", which means leading and participating in the continuing development of the Web and its standards. The new generation of Web technologies that is currently being specified by the W3C is intended to enhance the users' and publishers' control over the presentation of the information (e.g. through CSS), over the management of information (e.g. through RDF), and over its distribution (e.g. through P3P); based on technologies that structure and distribute data as objects, such as XML and HTTP-NG. Several areas will impact the use of technologies developed by the W3C in mobile environments.

The work of the W3C in this field is initiated and maintained through its Mobile Access Interest Group. Work which touches on the mobility of the user access devices is conducted in several areas, however. Specifically, the Mobile Access Interest Group is reviewing the HTML 4.0 specification for mobile aspects; investigating how user agent profile information can be managed within the W3C technology framework; and investigating position dependent information services. Due to the scope of mobile access, all areas of the W3C might concievably be involved in the future. Below, we point to some of those where mobile access may have an immediate impact.

4.1 XML and the next generation of HTML

The W3C has recently decided to initiate an activity to create a new generation of HTML. This will be based on XML, and is likely to include features that makes it more efficient for mobile use.

Meanwhile, other XML applications such as the Wireless Markup Language, WML, and the Synchronised Multimedia, SMIL, will continue to appear. These are likely to have components where mobile access will have an impact.

XML is also contiuing to develop into a full-featured system for information structuring.

More information about the HTML-NG activity

More information about XML

4.2 Vector Graphics

The W3C has created a Working Group, which will be chartered to produce a specification for a Scalable Vector Graphics format, written as a modular XML tagset and usable as an XML namespace, which can be widely implemented in browsers and authoring tools and which is suitable for widespread adoption by the content authoring community as a replacement for many current uses of raster graphics. For simple cases such as trivial inline graphics, it should be possible to hand author the SVG format, and it should be possible to cut and paste SVG graphical objects between documents and preserve their appearance, linking behavior and style.

This will mean that the graphics in Web documents will be smaller, faster, more interactive, and be displayable on a wider range of device resolutions from small mobile devices through office computer monitors to high resolution printers.

More information on the working group

4.3 Document Object Model and Formatting Model

In the presentation model for the new generation of web technologies, the formatting of a document is conducted through the use of a style sheet. This is a separate document which allows authors and users to attach style (e.g., fonts, spacing, and aural cues) to structured documents (e.g., HTML documents and XML applications). By separating the presentation style of documents from the content of documents, CSS2 and XSL simplifies Web and XML authoring and site maintenance. Local processing of a document might in the future also be conducted using a similar technology, called action sheets. Style sheets can have media-specific properties, which

makes them a possible candidate for use with mobile devices.

More information on style sheets.

The programmatical handling of a document is defined in the Document Object Model, a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them.

More information on the Document Object Model.

4.4 HTTP-NG

The purpose of the HTTP-NG activity is to design, implement, and test a new architecture for the HTTP protocol, based on a simple, extensible, distributed object-oriented model. This includes a protocol for the management of the network connections (WEBMUX), a protocol for transmitting messages between systems (WIRE), a set of methods, interfaces and objects that demonstrates a classical Web browsing case, as an example of what is possible with the new protocol; and a test bed to test the implementation.

More information on the HTTP-NG activity.

4.5 Accessibility

Accessibility for people with disabilities is relevant for mobile wireless devices as this is a potentially large marketplace (over 10% of the population), and in some cases accessibility is required (e.g. for sales in the US, under Section 255 of the US Telecommunications Act). In addition, functions, such as speech input or output, required to accommodate different kinds of disability have carry-over benefits for non-disabled

users of mobile devices, who may be using the devices in "hands-free" or "eyes-free" situations.

W3C's Web Accessibility Initiative (WAI) in coordination with other organizations is addressing Web accessibility through several areas of work; two of these, technology and guidelines, relate to mobile wireless devices.

In the area of technology, WAI liaises with W3C Working Groups developing technologies which can facilitate accessibility, such as HTML, CSS, SMIL, SVG. In the area of guidelines, WAI is developing guidelines for accessible page authoring, for user agents, and for authoring tools, and coordinating with the development of guidelines by the Mobile Access Interest Group.

More information about the Web Accessibility Initiative.

4.6 Internationalization

The correct representations of characters is an issue in all formats of writing, not just the Latin alphabet. The aim of this activity is for the World Wide Web to live up to its name, and the W3C continues work on the internationalization of the Web, with the aim of ensuring that the necessary features are included in W3C protocols and data format recommendations. The general goal of W3C's work on internationalization is to ensure that W3C's formats and protocols are usable world wide in all languages and writing systems.

More information about the internationalisation activity.

4.7 Metadata, Digital Signatures, and Trust Management

Our current focus is, broadly, on establishing trust in the new medium of the Web. This is a difficult problem, involving both social and technical issues. Trust is established through a complex and ill-understood social mechanism including relationships, social norms, laws, regulations,

traditions, and track records. Our activities are chosen to focus on specific areas that are both important and tractable.

There is a core of technical issues that are required in any system that is to be trusted:

The ability to make statements that have agreed-upon meanings.

> The W3C Metadata Activity provides a means to create machine-readable statements.

The ability to know who made the statement and to be assured that the statement is really theirs.

> The W3C Digital Signature Initiative provides a mechanism for signing metadata in order to establish who is making the machine-readable statement.

The ability to establish rules that permit actions to be taken, based on the statements and a relationship to those who made the statements.

> The PICS Rules specification allows rules to be written down so they can be understood by machines and exchanged by users.

The ability to negotiate binding terms and conditions.

> The now-completed JEPI project created the Protocol Extension Protocol (PEP) to provide for negotiation on the Web. Negotiation is also at the core of the Platform for Privacy Preferences Project (P3P).

Electronic commerce markup and payment.

> Currently, the W3C has two working groups in this field, on markup for electronic commerce, and for payment initiation.

More information on the metadata activity

5. Future Development From the WAP Forum in this Field

The WAP Forum's exclusive focus is mobile wireless technologies. The goal of WAP is to create recommendations and specifications that support the creation of advanced services on wireless devices, with particular emphasis

on the mobile telephone. The WAP Forum is creating recommendations and technologies which enable these services on all mobile devices and on all networks.

The WAP Forum has undertaken a variety of technical specification work relevant to the W3C/WAP Forum collaborative efforts. These efforts all relate to the use of World Wide Web technology on mobile devices, and ensuring that the quality of these services is sufficient for mass deployment.

5.1 Achieving the Mobile Wireless Web

WAP is focused on enabling the interconnection of the Web and wireless terminals. Significant focus has been given to mobile telephones and pagers, but all technology has been developed with broader applicability in mind. The goal of WAP is to enable an extremely wide range of wireless terminals, range from mass-market mobile telephones and pagers to more powerful devices, to enjoy the benefits of Web technology and interconnection.

Mobile devices have a unique set of features which must be exposed into the Web, in order to enable the creation of advanced telephony services, including:

Location-based services
Intelligent network functionality, including integration into the voice network
Voice/data integration

The WAP Forum is actively exploring solutions and specifications in these areas. Future technical work will address the ongoing evolution of wireless networks and mobile communication devices, including such issues as the integration of Third Generation wireless networking technology.

5.2 Bandwidth Efficiency

The WAP Forum is working to increase the bandwidth efficiency of Web technology, to make it more applicable to the wireless environment. WAP Forum work includes:

Smart Web Proxies
- proxies capable of performing intelligent transformation of protocols and content, enabling more efficient use of the network, adaptation to device characteristics and adaptation to network characteristics.

Efficient Content Encoding
- bandwidth efficient encodings of standard Web data formats such as XML.

Efficient Protocols
- bandwidth efficient adaptations of standard web protocols, such as HTTP.

5.3 Latency Constraints

The WAP Forum is working to improve the behavior of Web technology in the face of high network latencies, and in particular is focusing on the problems of:

Tuning network protocols
to be adaptive and efficient given wide ranging latencies.
Creating Web applications
which are resilient to either high latency environments or highly variable latency situations.

WAP Forum work in this area includes:

User agent state management
Protocol design (e.g., session state, fast session resumption, etc.)

5.4 Content Scalability

Mobile wireless devices are characterized by a different set of user interface constraints than a personal computer. The WAP Forum work in this area includes:

Content adaptation
- mechanisms allowing a Web application to adapt gracefully to the characteristics of the device (beyond the current HTTP/1.1 content negotiation model).

User interface scalability
- content formats, e.g., markup and display languages, that are suitable to improverished devices, but which scale well to more sophisticated devices.

6. Conclusion

In the area of Web technologies, the focus of the WAP Forum and the W3C overlap to a significant degree. Direct overlaps occur in the areas of intelligent proxies and protocol design; of XML applications; and in content adaptation, e.g., through the use of vector graphics and style sheets. Future cooperation may also occur in the area of electronic payment, where the work of the two groups has potential overlap.

Instead of developing diverging solutions, it is the intent of both groups to find common solutions that will address mobile requirements. In the area of web technology, our goals overlap, especially in the long run, allowing significant cooperation and shared development. To avoid fragmentation of the Web standards, the groups should cooperate, and focus on achieving the seamless integration of mobile devices into the Web.

Bruce Martin, WAP/Unwired Planet, September 23

Johan Hjelm, W3C/Ericsson, September 28

Active Server Pages: Using State To Your Advantage

Introduction

To date, we've talked about the infrastructure of Active Server Pages (ASP) and its potential with respect to developing large-scale, enterprise applications. So far, we've seen the capabilities of the Request and Response objects and briefly talked about other objects intrinsic to the ASP runtime environment. Coverage of this material was undoubtedly lengthy; however, I felt compelled to give you a thorough introduction to the topic preparing you for some of the more powerful features of ASP we would cover in articles to come.

In this article, I'll explore strategies for managing transient application state using ASP. Often referred to as simply 'state management', the concept of maintaining data for a specific user or application over a series of HTTP requests often poses some interesting problems related to web application development.

I'll introduce a new object in the ASP object model: the Session object. You'll use the **Session** object to manage information for individual users of a particular application. In a later installment, we'll use another object, the **Application** object to store common information that can be shared between all users of a single ASP application.

I'll also discuss state management techniques for ensuring that your ASP application scales gracefully in a distributed server environment. Distributed environments may be of special interest given that a single client requesting resources from your site may actually be serviced by a number of independent web servers making it harder to store information on a particular server about a particular client. The concept of load balancing HTTP requests and server distribution may become more important as your transaction requirements increase and you venture into developing enterprise-sized OLTP applications.

The 'State' of HTTP

Applications can be 'stateless' or 'stateful'. HTTP is a 'stateless' protocol. We'll talk more about 'stateful' later, but let it suffice to say that HTTP does not attempt to retain information, or state, about the current connection after a request completes. In a previous article, I introduced you to some of the characteristics of HTTP especially the request/response nature of interactions between clients (browsers) and servers.

In that article, we learned that when a browser needs to obtain a resource, it sends the server an HTTP Request message containing various pieces of information telling the server the nature of the request and, most importantly, the resource or document that it needs. When the server receives and processes this request, it generates an HTTP Response message containing, in some form or fashion, the contents of the requested document.

Once this sequence of events completes the connection terminates and the server waits for the next request. This request/response interaction allows a web server to process a large number of distinct requests and effectively service a large number of users without the overhead of maintaining long-lived information about each connection made.

Web Applications and State

We introduce complications into this scenario when information needs to survive or be recorded between several requests. It is not uncommon to encounter the same state management requirements in web applications as we do in conventional client-server applications - it's just harder to implement at times because the rules are slightly different.

In a typical application, a user 'logs in' to the system, performs a variety of functions, then 'logs out', effectively completing a 'session' with the application. For purposes of this discussion, a 'session' is the continuous usage of an application by the same user for a period of time. From a web application perspective, however, the concept of a potentially long-running 'session' (a persistent connection) does not exist by default, it must be simulated, in effect giving continuity to a series of independent pages.

ASP Sessions and the Session Object

To implement consistent user sessions on the Web, ASP provides a flexible solution that requires no special programming. The Session object, one of the intrinsic objects supported by ASP, provides the developer with a means of storing and retrieving data as well as performing session-related functions.

ASP automatically creates a Session object when a browser requests a page from an application that does not already have a session. When ASP creates a Session object, the user's 'session' begins. By default, the session lasts for twenty minutes, if no activity occurs within that timeframe, ASP destroys the session. This period can be adjusted by setting the Timeout property of the Session object.

Session Object Properties and Methods

The following table lists the properties and methods of the Session object:

Property	**Description**
SessionID	Returns the session identification for this user.
TimeOut	The timeout period for the session state for this application, in minutes.
Methods	**Description**
Abandon	This method destroys a **Session** object and releases its resources

Storing Information in the Session Object

An interesting feature of the Session object is that it supports a dynamic associative array that can be used to store named instances of data. This data can be simple data types, scalar variables and object references. A script simply assigns a value to a named entry in the array:

```
Session("User")="Jules"
```

The preceding example stores the string "Jules" in the Session object with the identifying name "User." Likewise, object references are stored using the VBScript keyword SET:

```
Set Session("PR") = Server.CreateObject("PR.Calculate")
```

Value can be retrieved from the Session object by referencing the Session object by name, as in the following:

```
Welcome <% = Session("User") %>!
```

Or

```
<% Response.Write Session("PR").Name %>
```

Note that before you store an object reference in the Session object, you should know what threading model it uses. Only objects marked as both free- and apartment-threaded can be stored in the Session object without locking the session to a single thread.

How ASP Keeps Track

The Session object maintains these name/value pairs for the lifetime of a user's logical session. For each ASP page requested by a user, the Session object preserves the information in the server's memory.

To locate this information, ASP uses a unique session identifier to match user requests with the information specific to that user's session. Session identifiers are simply globally unique identifiers (GUIDs) generated by ASP using the Windows API.

Session State and Cookies

ASP uses HTTP cookies to store and maintain Session Ids in the context of the client browser. For example, an ASP application with an established session would include a Set-Cookie header in the HTTP Response such as:

```
Set-Cookie: ASPSESSIONID=AXXQGHSSWQAJPUYT; path=/MyASPApp
```

The browser returns this cookie with each request made to the virtual web application directory /MyASPApp, giving ASP the opportunity to find the session-specific data previously stored.

It's important to note that the Session ID cookie contains no expiration time; therefore, it is only valid as long as the browser remains active.

The Session Object: Limitations

Given our discussion so far, the session object allows a relatively easy method of storing session specific information in the context of an ASP application. However, as you might have already guessed, a potential downside to this approach is that you cannot guarantee that a browser will accept cookies, the primary vehicle for identifying sessions via the Session ID, rendering the Session metaphor completely useless.

Furthermore, in more complex applications, if you are attempting to use a clustering strategy that allows multiple requests to be serviced by a number of different servers, the concept of session state stored on a particular server tends to break down in a hurry.

At this point, it's probably evident that the Session object has its' place in ASP application development, but there are definitely some limitations.

Another Method: A Database Table

There are times when the Session object may not be robust enough for your application needs. Although you may still choose to use the Session object for some non-critical pieces of data, your application may benefit from using an external storage mechanism such as relational database server to store state information.

When you think of it, a database server provides a very suitable model for storing an application's state. For one, the database can reside anywhere on the network and be accessible from any number of servers. Also, database servers today offer, at least, some performance enhancement features, referential integrity and even redundancy. For the most part, utilizing this storage medium to store transient application state is no more complex than any other database application.

The main complication arises when you try to decide how to uniquely identify and store application data and ensure that the data is always associated with the correct user on the network.

Choosing the Right Identity

There are several ways to uniquely identify a client making requests of your web server. At first glance, you might choose to use the client's IP address because every host on the Internet is assigned a unique IP address.

This strategy tends to weaken when you think of the numerous IP addresses dynamically assigned by Internet Service Providers that probably expire in a couple of hours. Better yet, if the client is connecting via a proxy server, they could be one of a hundred clients masquerading behind the proxy's single IP address.

A strategy you may choose involves managing the unique identifier yourself, in effect, taking over the management of the session and the unique identifier that identifies its' data. This probably doesn't alleviate the need for storing this unique identifier in a client-based cookie, however, using C++ or VB5 and the Windows API, you too can create unique identifiers.

Click here to see an example of how to implement this unique identifier using VB5 and the Windows API.

Self-Managed Sessions

Once identifying the session is no longer a problem, you need to implement the code to persist the session state.

You may choose one of several methods:

First, you could write a generic distributed Session object as a COM component that behaved much like the intrinsic Session object provided by ASP. We haven't covered COM components in this series of articles yet, however, if you are familiar with the concept you probably get the idea of how the underlying database would be managed. A database table for managing this state might look something like this:

Name	Type	Description
SessionID	Character	Unique session identifier generated by your algorithm.
PropertyName	Character	Name of property you wish to store for this session.
Value	Character	Value of property you wish to store for this session.

Given a Session ID and the desired property name, you could either insert a new value or retrieve an existing value from the table. As long as property names are kept unique and data storage needs are limited to simple data types, you have essentially created a custom version of the ASP Session object.

On the other hand, if you are interested in only a finite amount of data, your table structure could be much more straightforward. Consider the following, very specialized adaptation of this solution, that stores only limited demographic information for the current client:

Name	Type	Description
SessionID	Character	Unique session identifier generated by your algorithm.
Name	Character	Person Name
Address	Character	Person Address

What Happens When It's All Over?

If you choose to store state information in an external table, one final complication may be garbage collection. In other words, when a session 'ends' what do you do with information previously recorded in the database?

A straightforward way to approach this in the self-managed session model, is to create your own methods of construction and destruction with respect to the

individual session.

For example, when you create a session for a user, your application is responsible for the destruction of that session via your custom Session object. This strategy obviously has a couple of holes in that users have a nasty habit of not always logging off.

Yet another strategy might be to 'age' the properties and sessions in your session management table and clean them up periodically. Another less-than-perfect, but useable solution if self-managed sessions is your choice.

A Final Approach: HTTP-based State

One last approach might be the simplest, but could also be the least reusable. You could encapsulate the state in the HTTP Request and Response messages.

Although this technique is certainly not new, and is not limited to ASP applications, HTML provides for encapsulating state information through the use of both normal, visible form fields and form fields having the HIDDEN attribute.

The basic concept is that given a known amount of state, you can store name/value pairs in form fields, basically eliminating the need for server-based application state.

Using this technique gives you the ability to use the standard HTML form submission model to create and encapsulate the HTTP Request and the ASP Response object to query property specific information. To complete the model, you have to carry the state received from the HTTP Request forward to the HTTP Response by regenerating the additional hidden fields and their respective values.

An example HTML form using this technique might initially look as follows.

```
<HTML>
<BODY>
<FORM ACTION="FIRST.ASP" METHOD=POST>
<INPUT TYPE=TEXT NAME="USERID">
<INPUT TYPE=PASSWORD NAME="PASSWORD">
<INPUT TYPE=SUBMIT NAME="MYSUBMIT" VALUE="OK">
</FORM>
</BODY>
</HTML>
```

The above example is a standard form requesting the user id and password with a submit button that invokes the server script FIRST.ASP. The Submit button 'packages' these fields into the HTTP request for transmission to the server.

When returning information to the client, our server side script generates the next data entry form preserving state from previous forms in hidden fields as follows:

```
<FORM ACTION="NEXT.ASP" METHOD=POST>
<INPUT TYPE=TEXT NAME="STREETADDRESS">
<INPUT TYPE=TEXT NAME="CITY">
<INPUT TYPE=HIDDEN NAME="USERID" VALUE="1234">
<INPUT TYPE=SUBMIT NAME="MYSUBMIT" VALUE="OK">
</FORM>
```

Notice the usage of both hidden and visible form fields. Using this technique, we preserved the USERID field across HTTP Requests, giving server-side script the ability to re-establish the state for this user based on the contents of USERID along with any other data we may have obtained with additional form fields.

Conclusions

The HTTP request/response interaction model has been instrumental in creating one of the most effective distributed processing platforms of all time: the World Wide Web. HTTP has its limitations and certainly its' 'stateless' nature forces us as software developers to sometimes think in a totally different paradigm spending most of our time dealing with the hassles of storing and retrieving data instead of pursuing the more interesting aspects of our application.

When all is said and done, you have a choice. In this article I've attempted to outline a couple of these choices for you at a reasonably high level and offer some potential architectural suggestions for those of you planning large-scale applications.

In summary, remember a couple of things...

For reasonably lightweight, single server applications (and product demos), the ASP Session object provides a simple solution for storing and retrieving transient application state information. If you keep in mind what you're attempting to store in this object and don't overuse it, you'll get by just fine.

For more complex applications, especially those with significant state management requirements and those that must maintain state across multiple servers and multiple applications you may need to investigate some of the alternatives we've discussed or formulate your own state management strategy.

As always, I appreciate having you along as my guest and would be even more appreciative of any feedback you may have. See you next time!

Author: Keith Cox
Date: 04/20/98

More articles about Active Server Pages
More articles by Keith Cox

Author Biography

body>